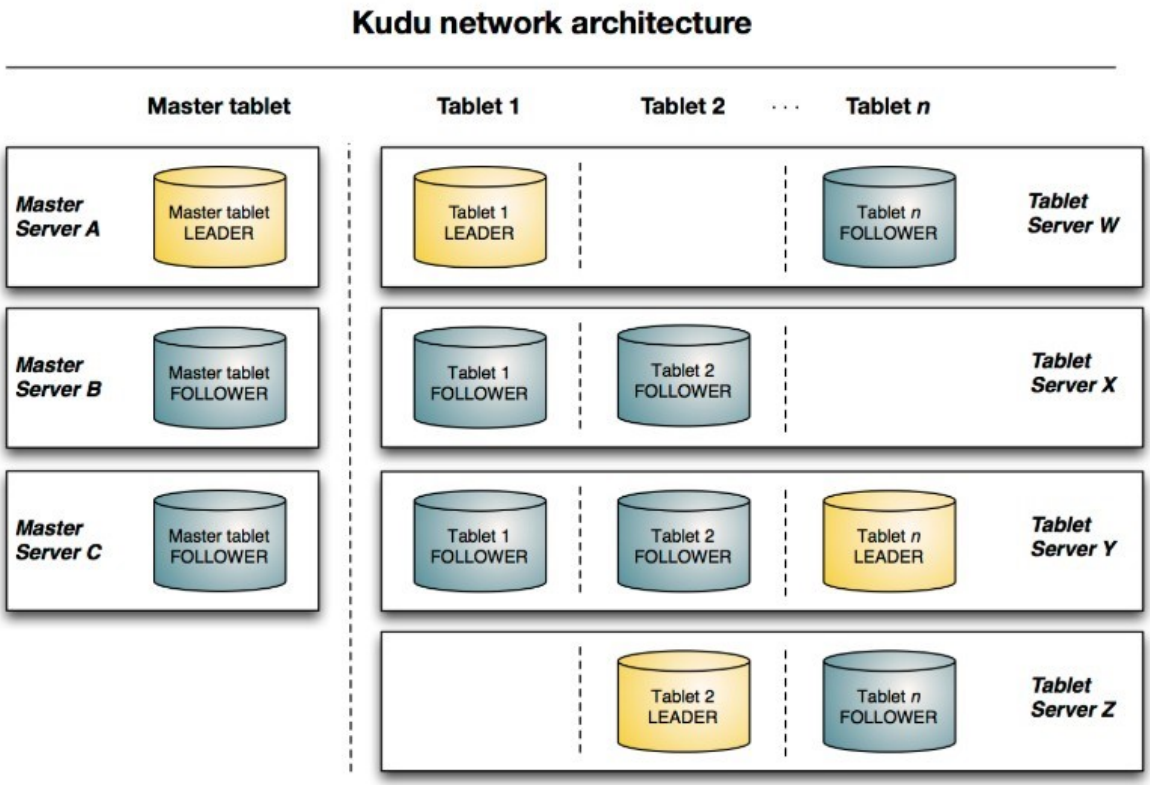
1. Kudu 高层架构及基本术语

1.1. 高层架构

高层架构图:



如上图，跟 HBase 类似，Kudu 是典型的主从架构。一个 Kudu 集群由主节点即 Master 和若干个从节点即 Tablet Server 组成。Master 负责管理集群的元数据（类似于 HBase 的 Master），Tablet Server 负责数据存储（类似 HBase 的 RegionServer）。在生产环境，一般部署多个 Master 实现高可用（奇数个、典型的是 3 个），Tablet Server 一般也是奇数个。

1.2. 核心术语

除了 Master 和 Tablet Server，Kudu 中还有很多术语，为便于理解统一整理理解一下。

1.2.1. Table 表

Kudu 中 Table（表）的概念跟其他关系型数据库一样，Table 是数据存储在 Kudu 的位置。表具有 schema（表结构）和全局有序的 primary key（主键）。table 被水平分成很多段，每个段称为 Tablet。

表（Table）是数据库中用来存储数据的对象，是有结构的数据集合。kudu 中的表具有 schema（纲要）和全局有序的 primary key（主键）。kudu 中一个 table 会被水平分成多个被称之为 tablet 的片段。

1.2.2. Tablet

一个 tablet 是 一张表 table 连续的 segment（片段），类似于 HBase 的 region 或关系型数据库 partition（分区）。每个 tablet 存储着一定连续 range 的数据（key），且 tablet 两两间的 range 不会重叠。一张表的所有 tablet 包含了这张表的所有 key 空间。

tablet 会冗余存储。放置到多个 tablet server上，并且在任何给定的时间点，其中一个副本被认为是 leader tablet，其余的被认之为 follower tablet。每个 tablet 都可以进行数据的读请求，但只有 Leader tablet 负责写数据请求。

1.2.3. Tablet Server

Tablet server 是 Kudu 集群中的从节点，负责数据存储，并提供数据读写服务。

一个 Tablet server 存储了 table 表的 tablet，向 kudu client 提供读取数据服务。对于给定的 tablet，一个 tablet server 充当 leader，其他 tablet server 充当该 tablet 的 follower 副本。

只有 leader 服务写请求，然而 leader 或 followers 为每个服务提供读请求 。一个 tablet server 可以服务多个 tablets ，并且一个 tablet 可以被多个 tablet servers 服务着。

1.2.4. Master

集群中的主节点，负责集群管理、元数据管理等功能。

保持跟踪所有的 tablets、tablet servers、catalog tables（目录表）和其它与集群相关的 metadata。在给定的时间点，只能有一个起作用的 Master（也就是 leader）。

如果当前的 leader 消失，则选举出一个新的 master，使用 Raft 协议来进行选举。master 还协调客户端的 metadata operations（元数据操作）。

例如，当创建新表时，客户端内部将请求发送给 master。 master 将新表的元数据写入 catalog table（目录表，元数据表），并协调在 tablet server 上创建 tablet 的过程。

所有 master 的元数据都存储在一个 tablet 中，可以复制到所有其他候选的 master。tablet server 以设定的间隔向 master 发出心跳（默认值为每秒一次）。

1.2.5. Raft Consensus Algorithm

Kudu 使用 Raft consensus algorithm 作为确保常规 tablet 和 master 数据的容错性和一致性的手段。通过 Raft 协议，tablet 的多个副本选举出 leader，它负责接受请求和复制数据写入到其他 follower 副本。一旦写入的数据在大多数副本中持久化后，就会向客户确认。给定的一组 N 副本（通常为 3 或 5 个）能够接受最多 (N - 1)/2 错误的副本的写入。

1.2.6. Catalog Table(目录表)

Catalog table 是 Kudu 的元数据表。它存储有关 tables 和 tablets 的信息。catalog table（目录表）不能被直接读写，它只能通过客户端 API 中公开的元数据操作访问。catalog table 存储以下两类元数据：

1、Tables：table schemas 表结构，locations 位置，states 状态

2、Tablets：现有 tablet 的列表，每个 tablet 的副本所在哪些 tablet server，tablet 的当前状态以及开始和结束的 keys（键）。

2. Kudu 中的相关概念和机制

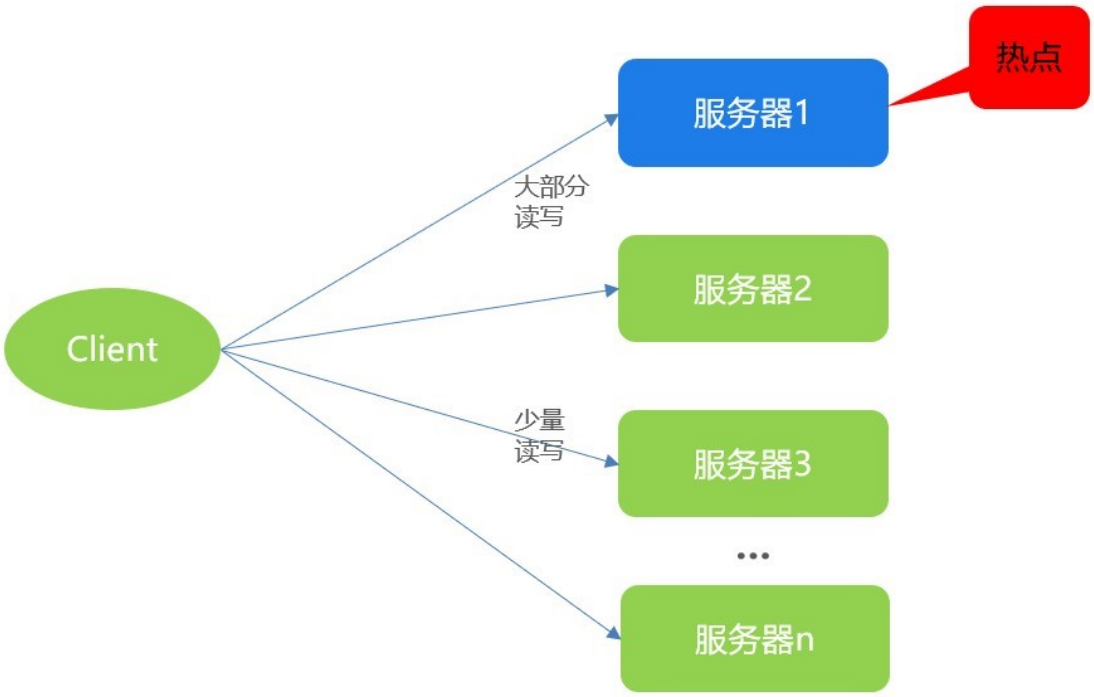
作为分布式存储，一个至关重要的事情就是把负载分散到多个服务器上（读写负载/数据分布），让集群中的每个节点都充分的且尽量均衡的参与读写操作，会大幅提高整体吞吐和降低延迟。那么 Kudu 通过什么机制实现数据最优分布？

2.1. 主键

Kudu 是有主键的，跟关系型数据库的主键是概念相通的。Kudu 基于主键来重组和索引数据，但是没有二级键、没有二级索引，数据存储到哪台服务器也跟主键密切相关。因此主键设计不好将直接影响查询性能，通常还会导致热点问题。

2.2. 热点问题

热点问题：熟悉 HBase 或者其他分布式存储的人对热点问题一定不陌生，任何一个分布式系统(无论存储还是计算)都无法回避这个问题。所谓热点问题(hotspotting)，就是在分布式系统中大多数读或者写请求落到一台或者几台服务器时，我们称之为热点。



热点问题产生的原因：热点问题多半是分区设计跟读写模式不匹配造成的，以车联网电池温度传感器上传数据为例：

温度传感器数据(假设传感器采频率相用)

date	sensorId	value(温度)
2019-01-01	A	20
2019-01-01	J	36
2019-01-02	Z	42

分区设计：~~时间范围分区(日期)~~

服务器1
分区1
从：2017/01/01
到：2017/12/31

服务器2
分区2
从：2018/01/01
到：2018/12/31

服务器3
分区3
从：2019/01/01
到：2019/12/31

写热点：
2018年2月13号当天，所有数据都会写入到分区2，分区1即服务器3就是写热点

读热点：
在业务上我们往往最关心最近你三个月的温度数据，那么绝大多数分析查询语句势必落到分区3，分区3及服务器3就是读热点

另外还可能造成存储热点和计算热点

因此要解决热点问题，必须考虑综合考虑具体业务在以下三方面的事情：

```
1、读操作模式(吞吐和延迟)
2、写操作模式(吞吐和延迟)
3、存储开销(压缩比)
```

2.3. 分区

Kudu 目前支持三种分区方式：

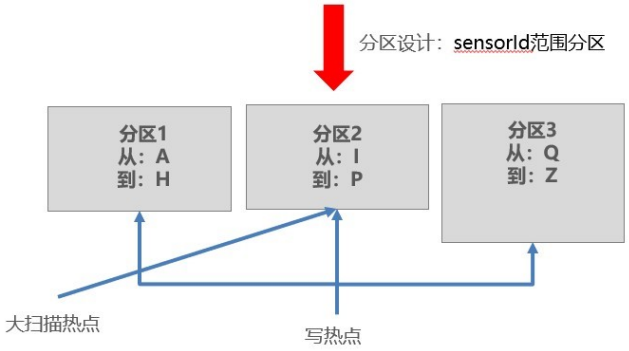
```
范围分区
哈希分区
多级分区
```

依然以温度传感器数据为例我们来看看各种分区。

2.3.1. 范围分区

温度传感器数据(假设传感器采频率相用)

date	sensorId	value(温度)
2019-01-01	A	20
2019-01-01	J	36
2019-01-02	Z	42

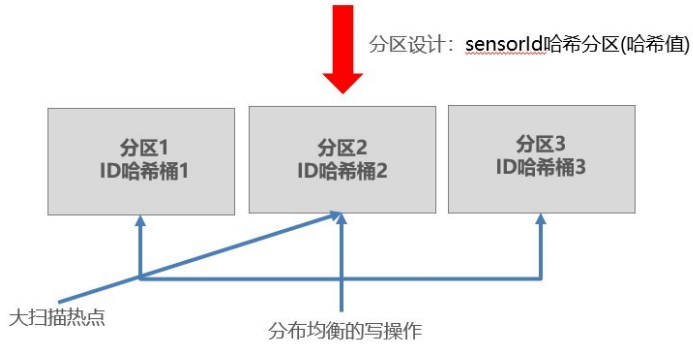


- 如果某个ID范围的传感器数量更多?
- 如果读取某个传感器的所有数据

2.3.2. 哈希分区

温度传感器数据(假设传感器采频率相用)

date	sensorId	value(温度)
2019-01-01	A	20
2019-01-01	J	36
2019-01-02	Z	42



- 如果某个ID范围的传感器数量更多?
- 如果读取某个传感器的所有数据

2.3.3. 多级分区

- 如果某个ID范围的传感器数量更多？
- 如果读取某个传感器的所有数据